

CMSC 411

Computer Architecture

Lecture 15

Handling Pipeline Hazards



Lecture's Overview

□ Previous Lecture:

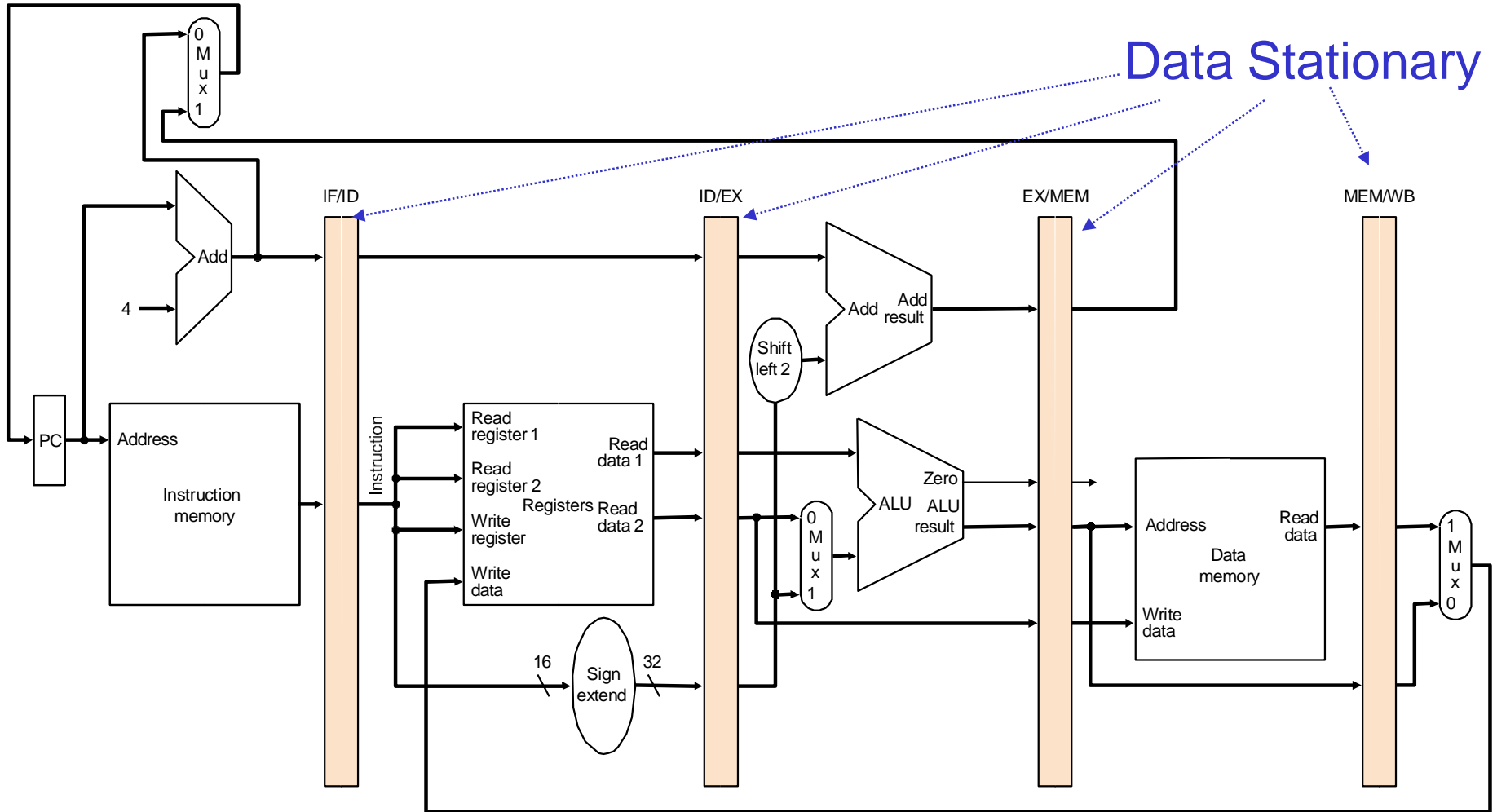
- Designing a pipelined datapath
 - Standardized multi-stage instruction execution
 - Unique resources per stage
- Controlling pipeline operations
 - Simplified stage-based control
 - Detailed working example

□ This Lecture:

- Pipeline hazard detection
- Handling hazard in the pipeline design
- Supporting exceptions



Pipelined Datapath



1

2

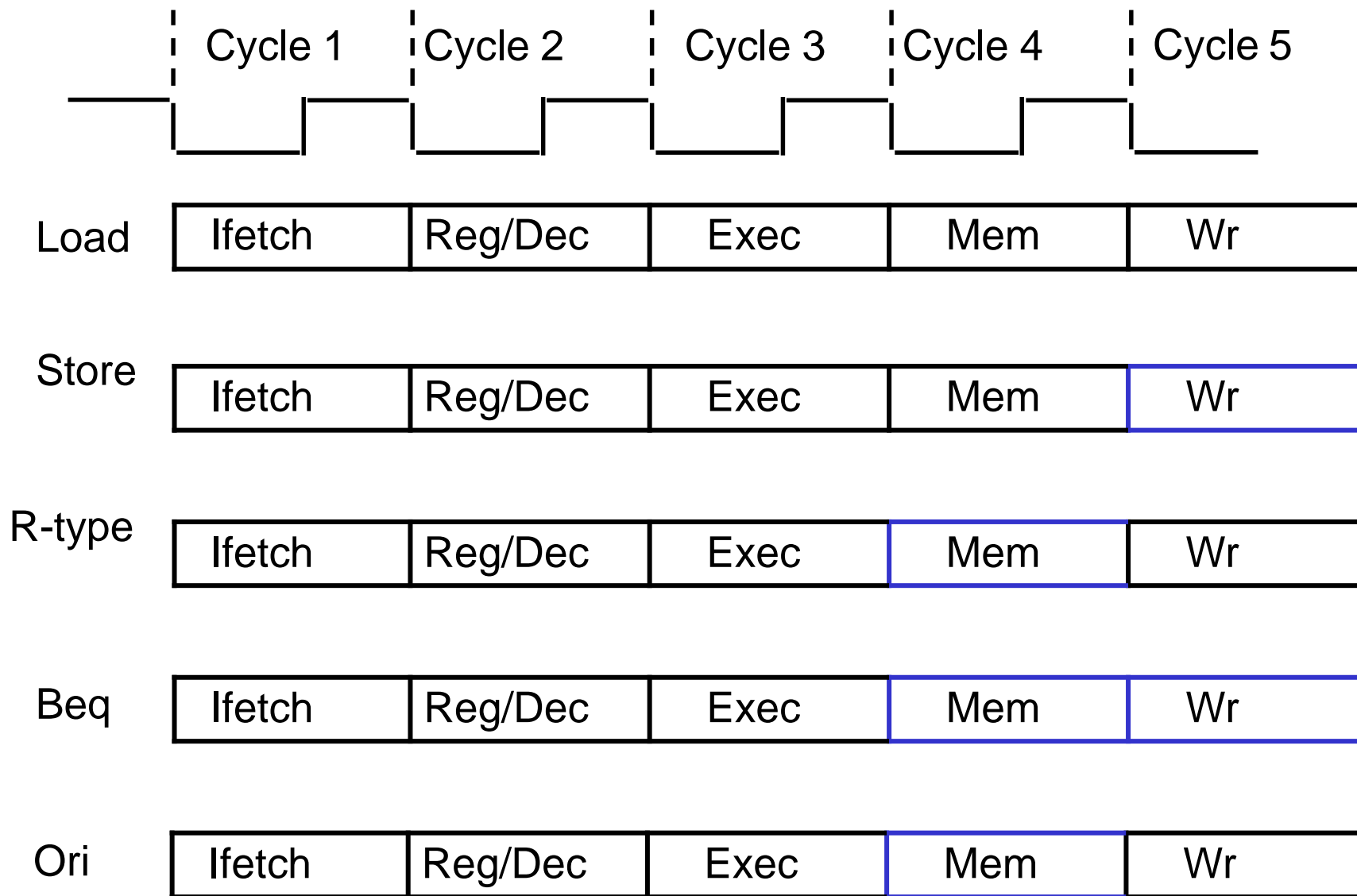
3

4

5



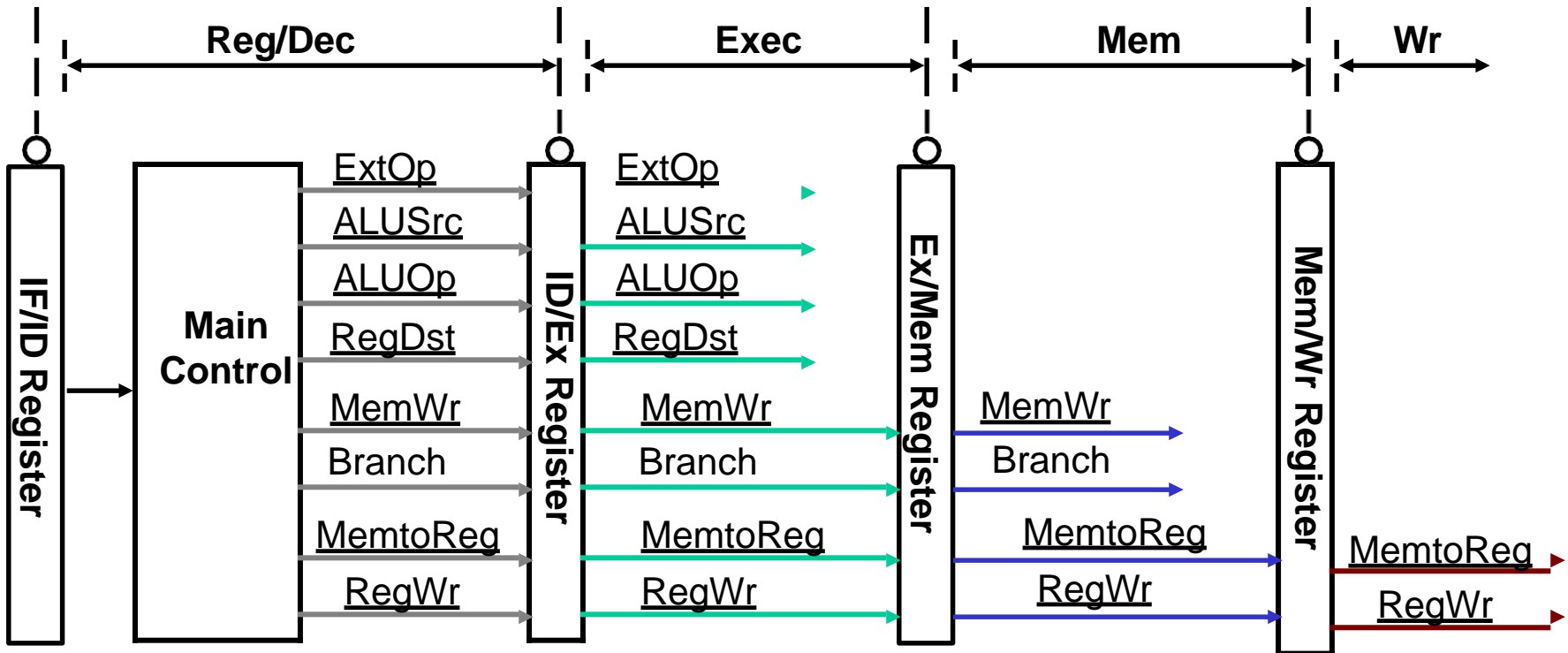
Standardized Five Stages Instructions



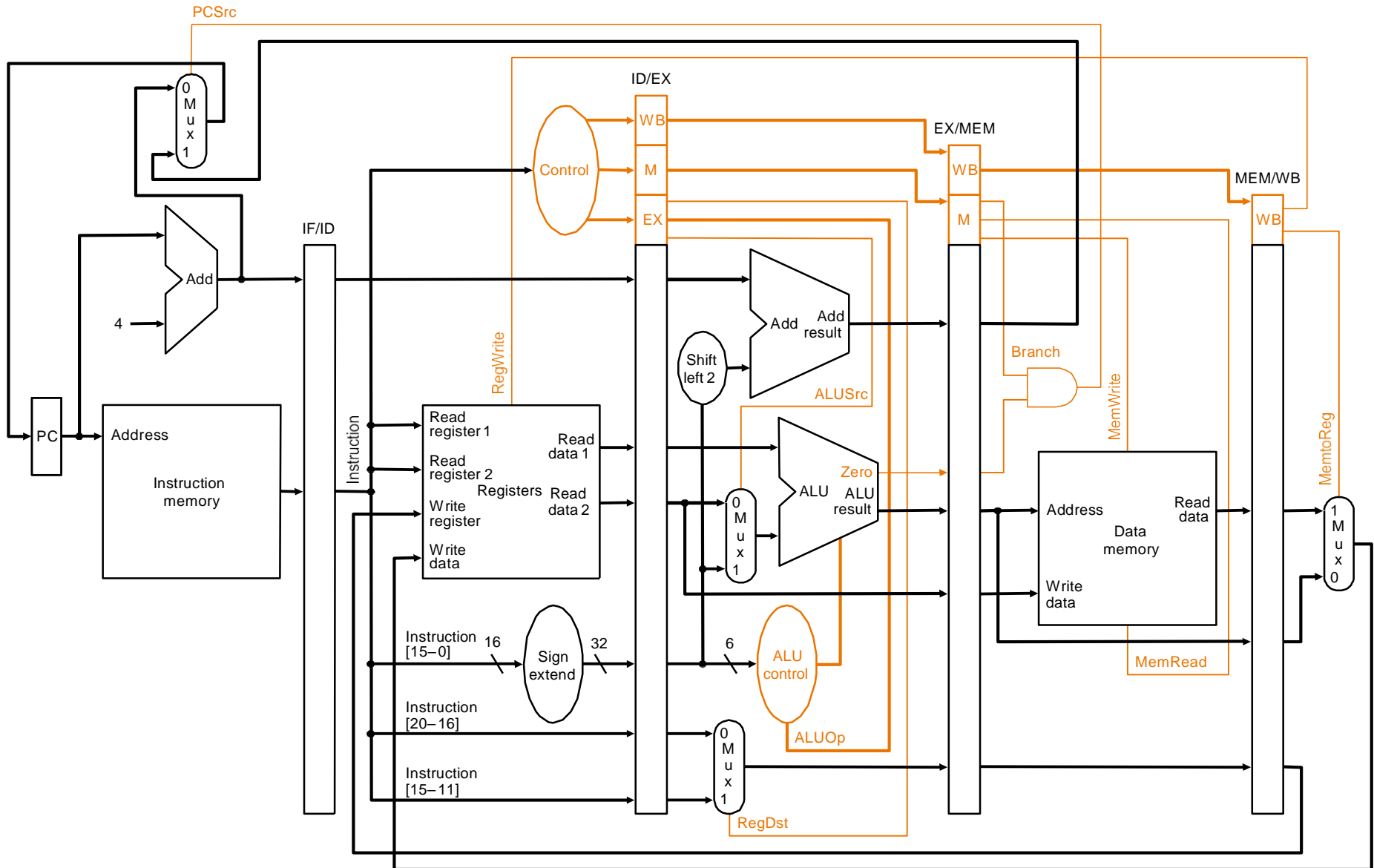
Data Stationary Control

The Main Control generates the control signals during Reg/Dec

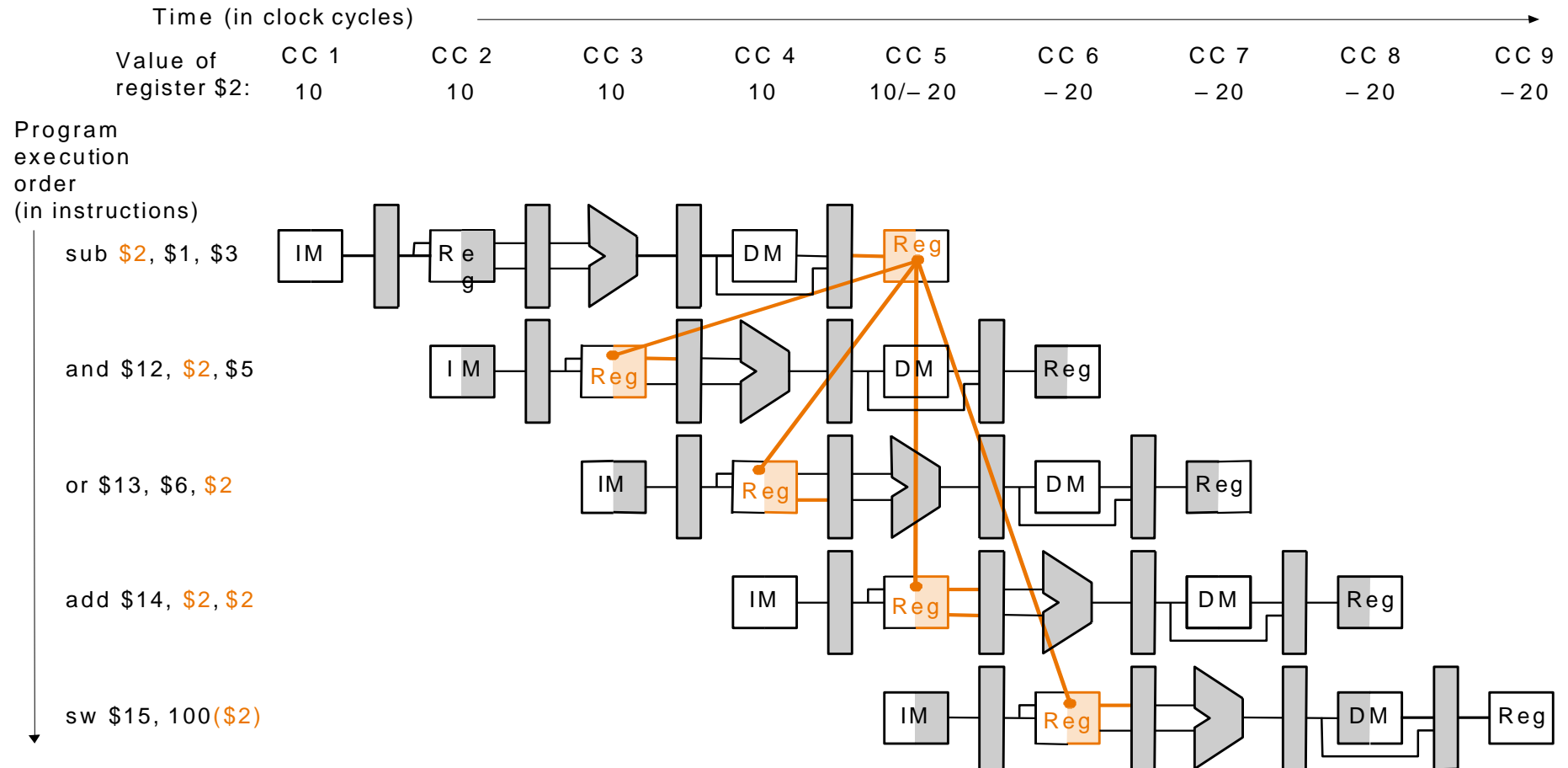
- ➔ Control signals for Exec (ExtOp, ALUSrc, ...) are used 1 cycle later
- ➔ Control signals for Mem (MemWr Branch) are used 2 cycles later
- ➔ Control signals for Wr (MemtoReg RegWr) are used 3 cycles later



Datapath + Data Stationary Control

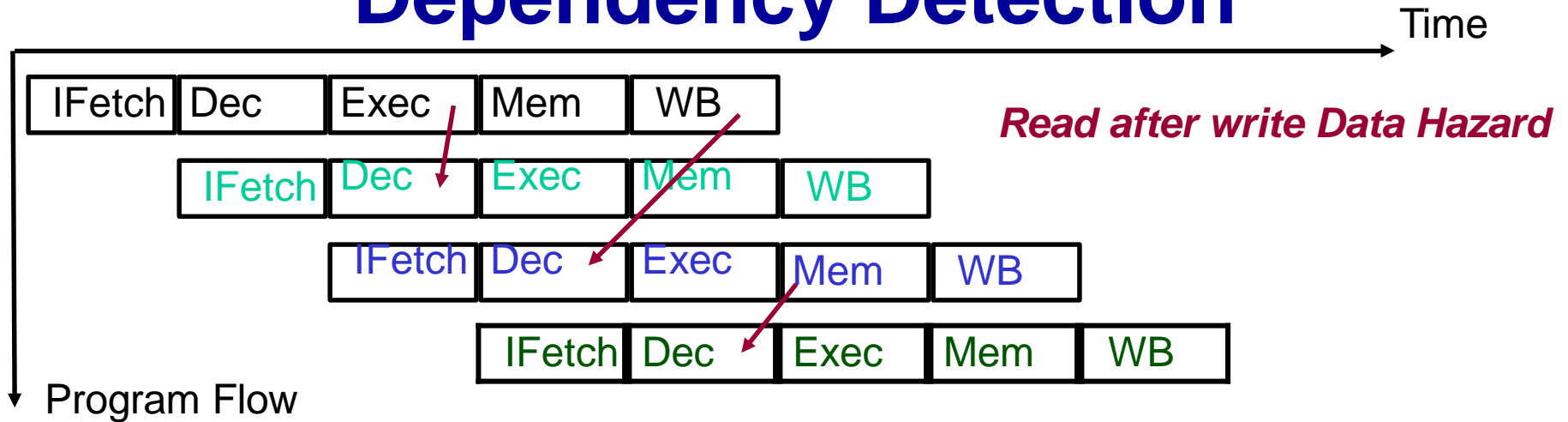


Data Hazards



- ❑ Generally caused by data dependence among instructions
- ❑ Can cause erroneous semantics if went undetected and avoided
- ❑ Raised when an instruction depends on result of a prior instruction still in the pipeline and attempts to use item before it is ready

Dependency Detection



- ❑ Data Hazards is caused by backward dependency at the DEC stage
- ❑ A hazardous register is to be updated at an earlier instruction at the EX, MEM or WB stages of the pipeline
- ❑ It is assumed that register writing takes precedence over reading if happened in the same clock cycle \Rightarrow **WB stages will not cause data hazard**

Data Hazard conditions:

1a. $EX/MEM.Register-Rd = ID/EX.Register-Rs$

1b. $EX/MEM.Register-Rd = ID/EX.Register-Rt$

2a. $MEM/WB.Register-Rd = ID/EX.Register-Rs$

2b. $MEM/WB.Register-Rd = ID/EX.Register-Rt$

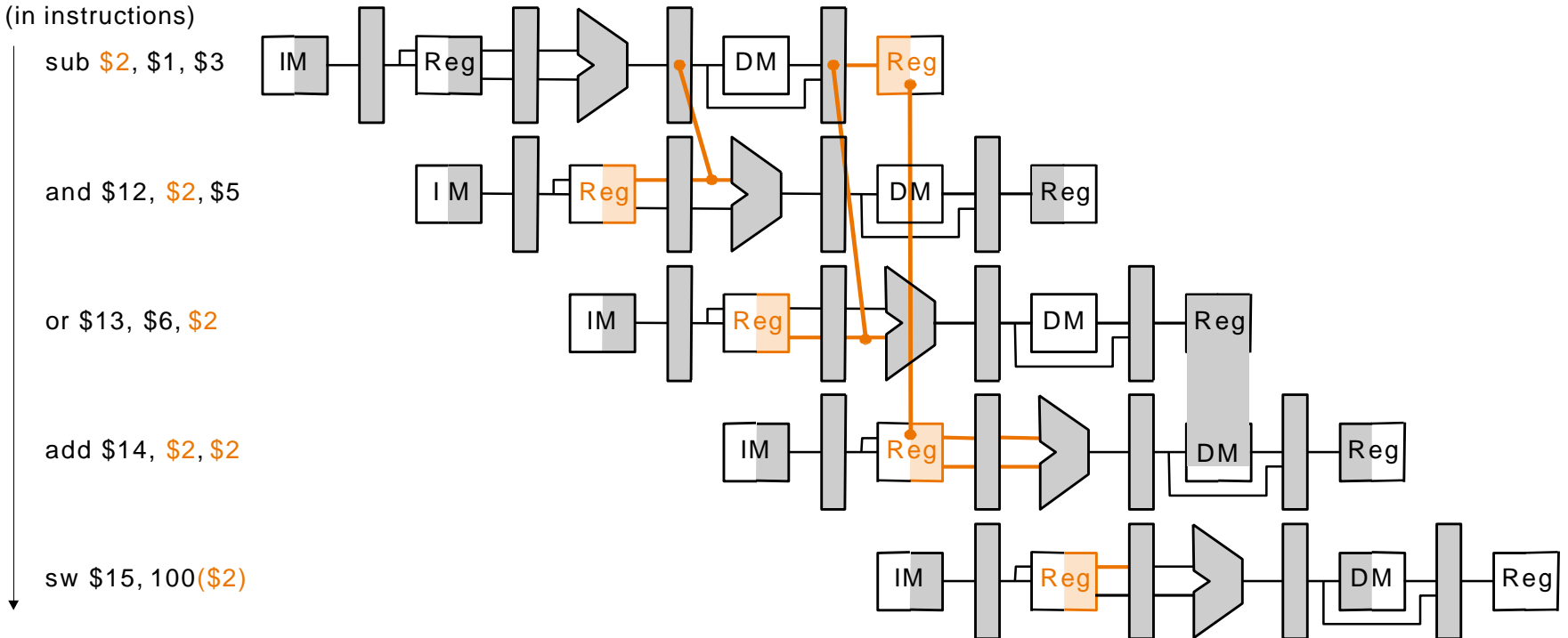


Data Forwarding

Time (in clock cycles) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register \$2 :	10	10	10	10	10/-20	-20	-20	-20	-20
Value of EX/MEM :	X	X	X	-20	X	X	X	X	X
Value of MEM/WB :	X	X	X	X	-20	X	X	X	X

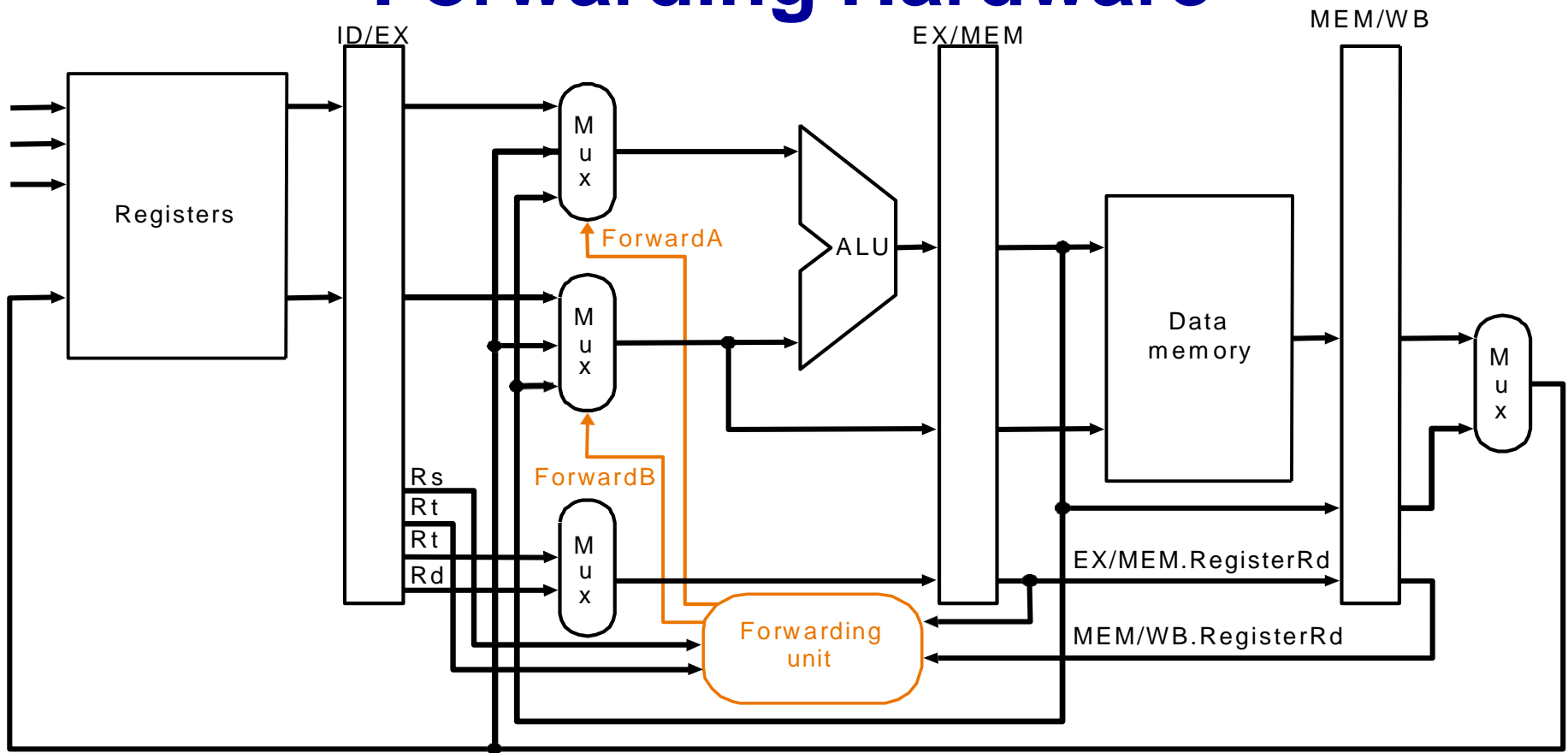
Program execution order (in instructions)



- Detecting data hazard conditions allows using intermediate results
- Forward only when there is a WB stage (check the RegWrite signal)



Forwarding Hardware



EX Hazard:

if (EX/MEM.RegWrite & (EX/MEM.Register-Rd = ID/EX.Register-Rs)) ForwardA = EX/MEM

if (EX/MEM.RegWrite & (EX/MEM.Register-Rd = ID/EX.Register-Rt)) ForwardB = EX/MEM

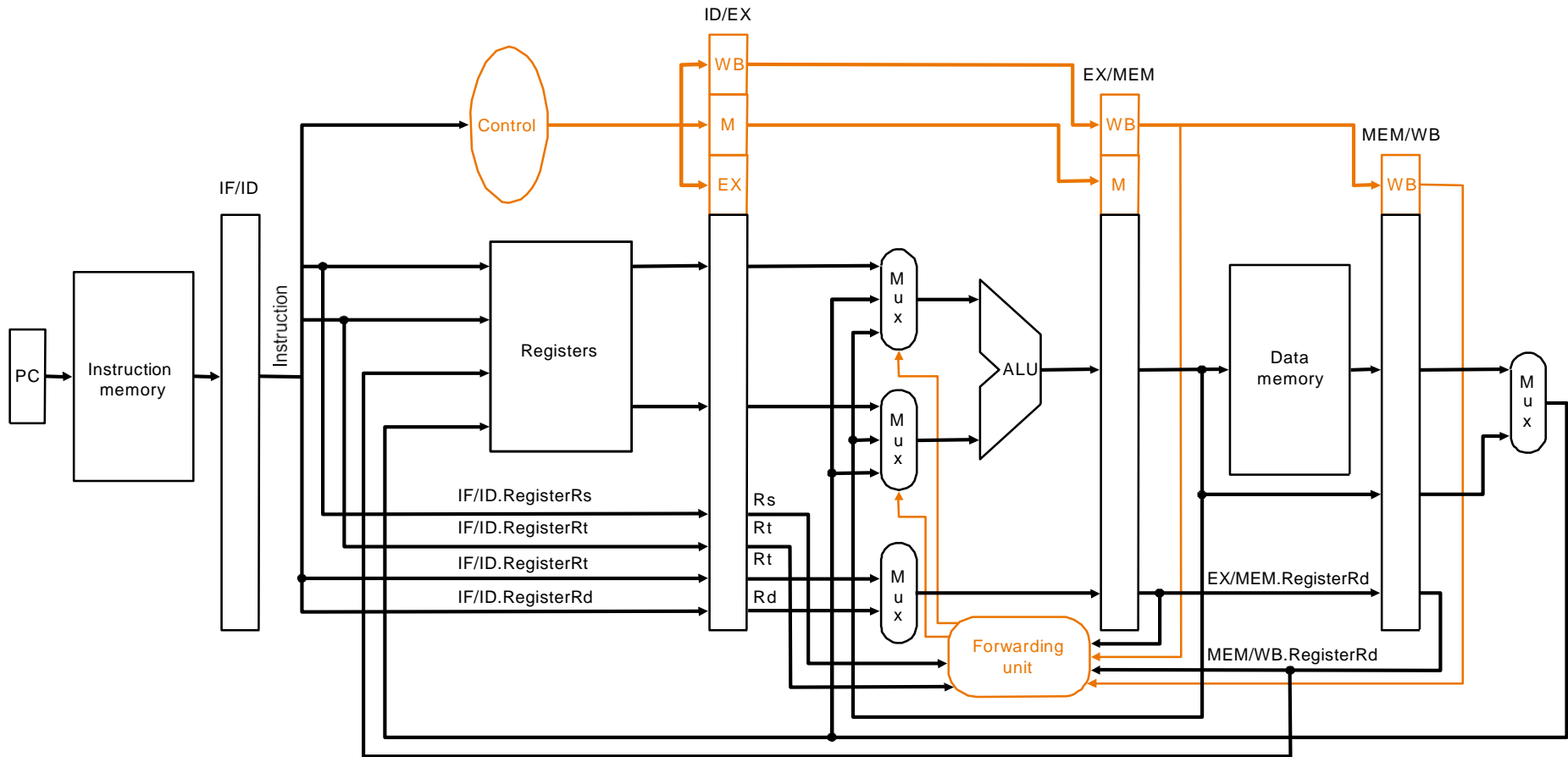
MEM Hazard:

if (MEM/WB.RegWrite & (MEM/WB.Register-Rd = ID/EX.Register-Rs)) ForwardA = MEM/WB

if (MEM/WB.RegWrite & (MEM/WB.Register-Rd = ID/EX.Register-Rt)) ForwardB = MEM/WB



Forwarding Register of Successive Use



Example:

add \$1, \$1, \$2;
 add \$1, \$1, \$3;
 add \$1, \$1, \$4;

if (MEM/WB.RegWrite & (EX/MEM.Register-Rd ≠ ID/EX.Register-Rt) &
 (MEM/WB.Register-Rd = ID/EX.Register-Rs)) ForwardA = MEM/WB
 if (MEM/WB.RegWrite & (EX/MEM.Register-Rd ≠ ID/EX.Register-Rs) &
 (MEM/WB.Register-Rd = ID/EX.Register-Rt)) ForwardB = MEM/WB



Data Forwarding Example

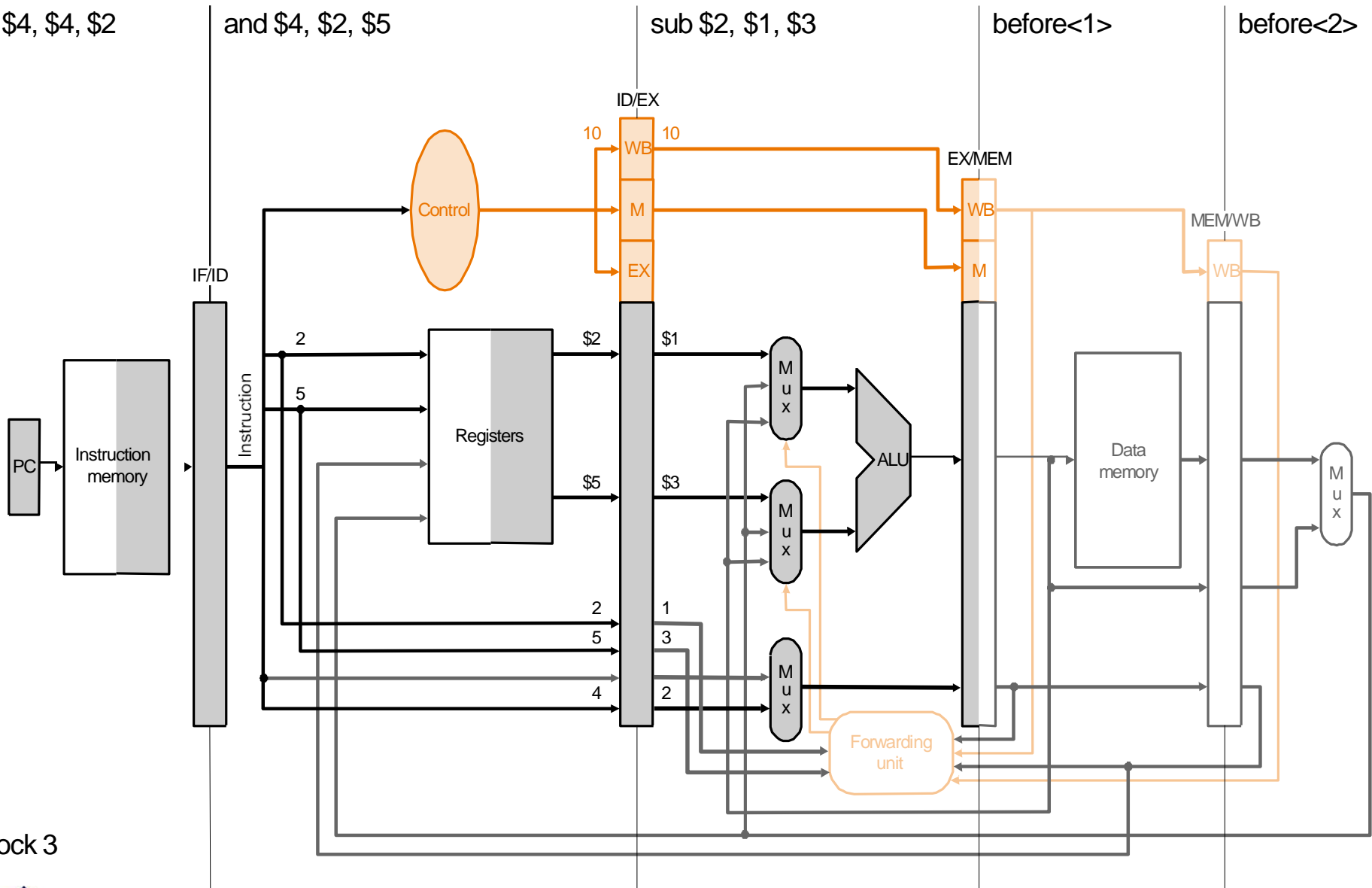
or \$4, \$4, \$2

and \$4, \$2, \$5

sub \$2, \$1, \$3

before<1>

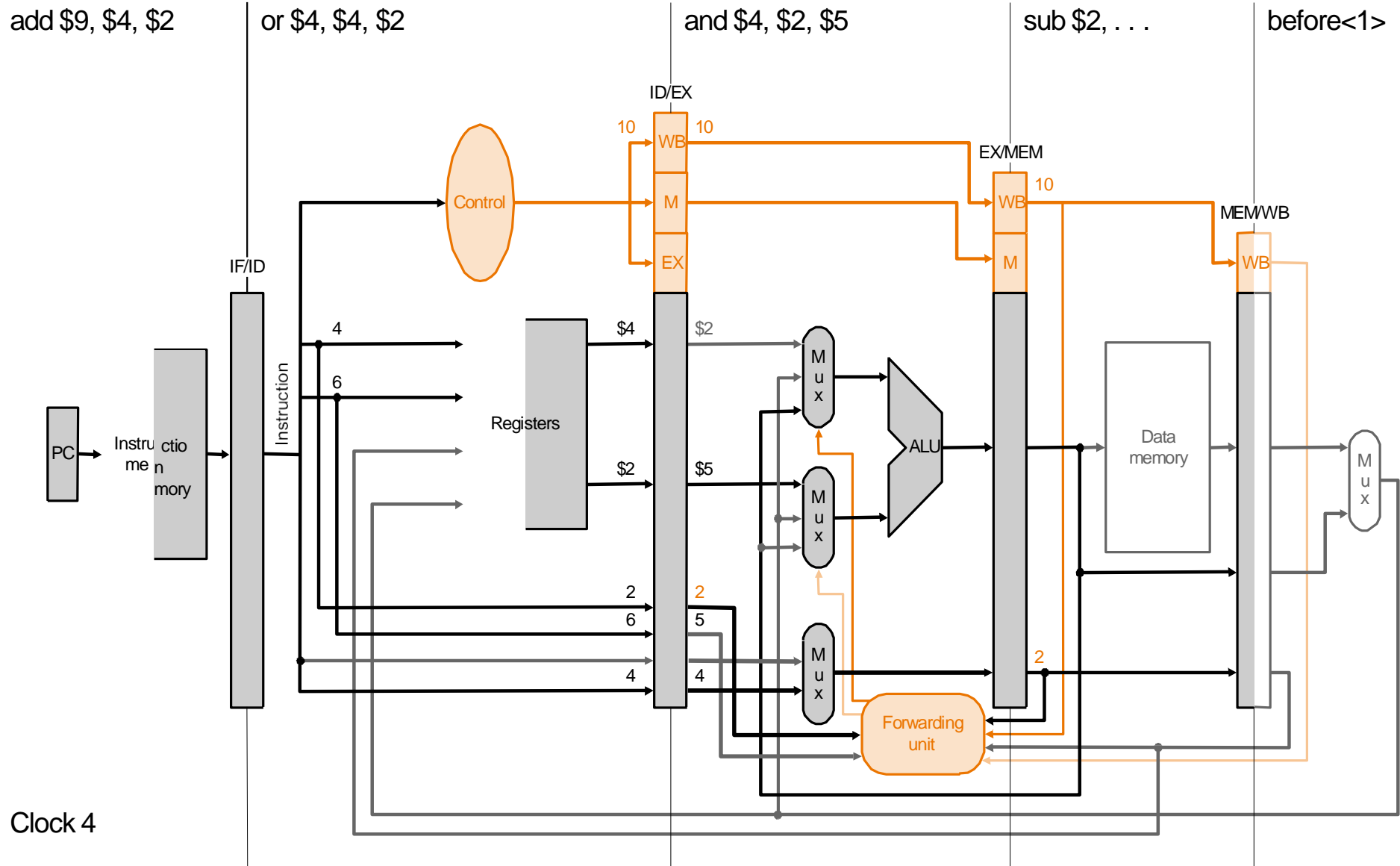
before<2>



Clock 3



Example



Clock 4



Example

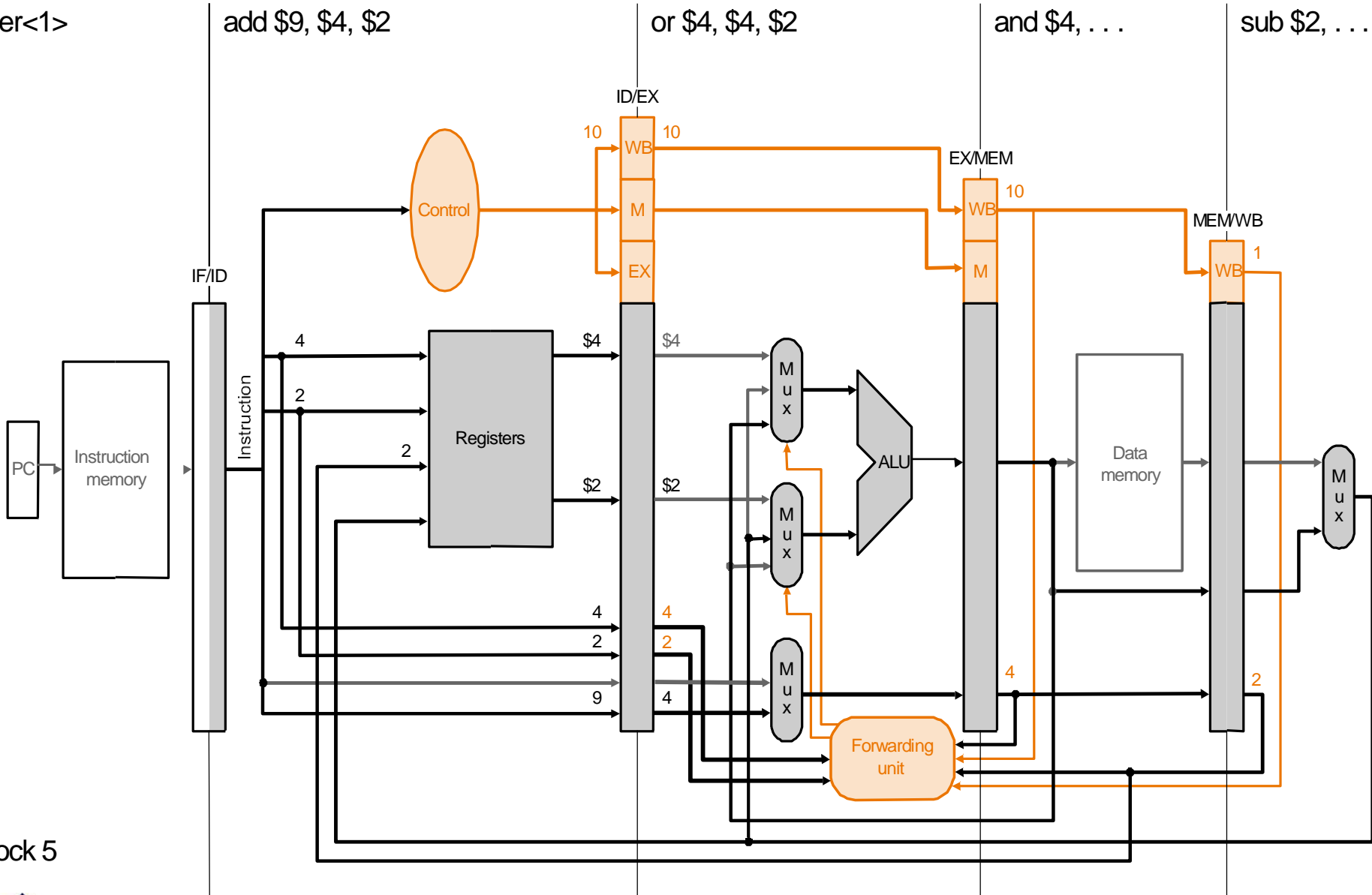
after<1>

add \$9, \$4, \$2

or \$4, \$4, \$2

and \$4, ...

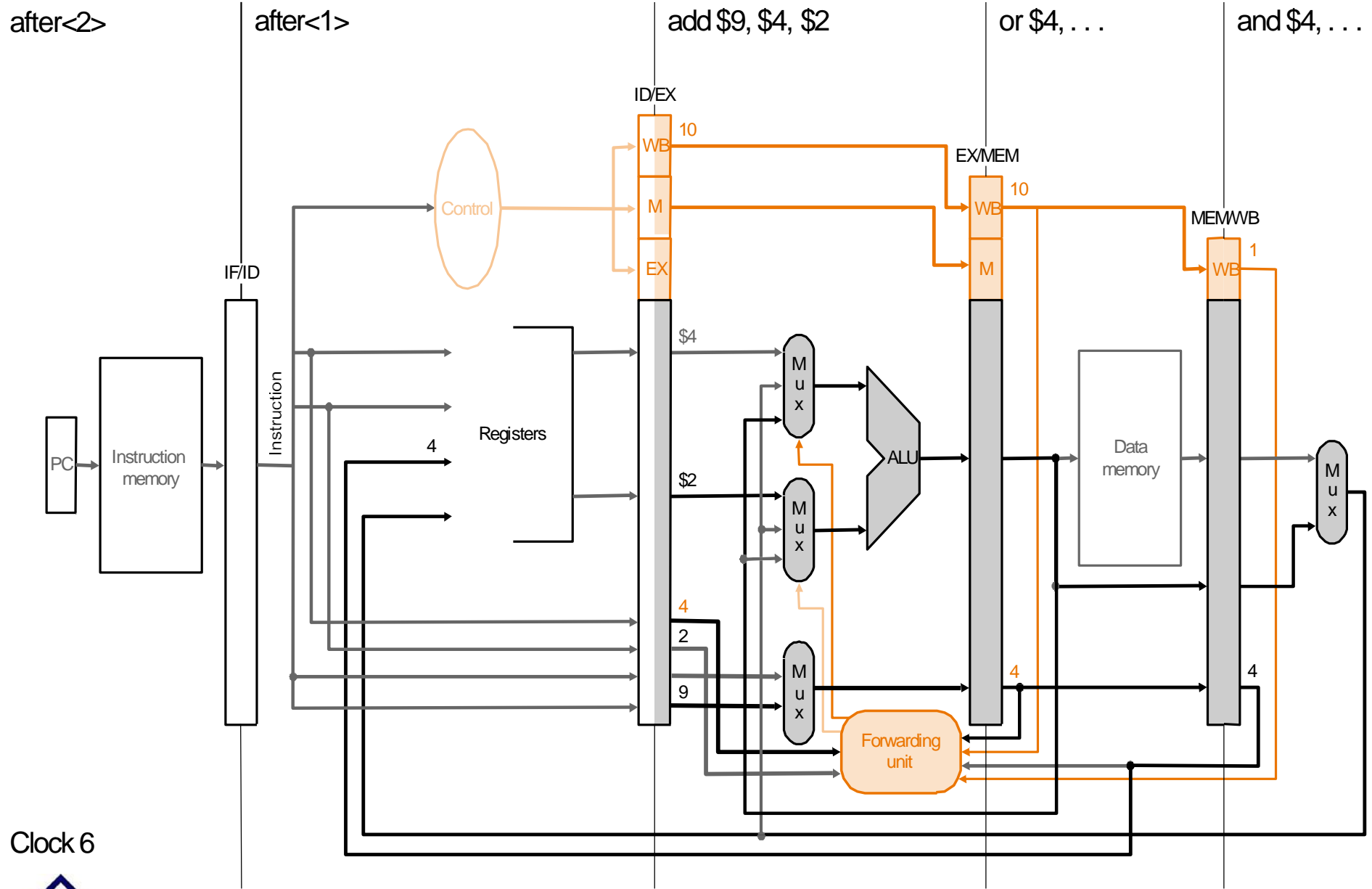
sub \$2, ...



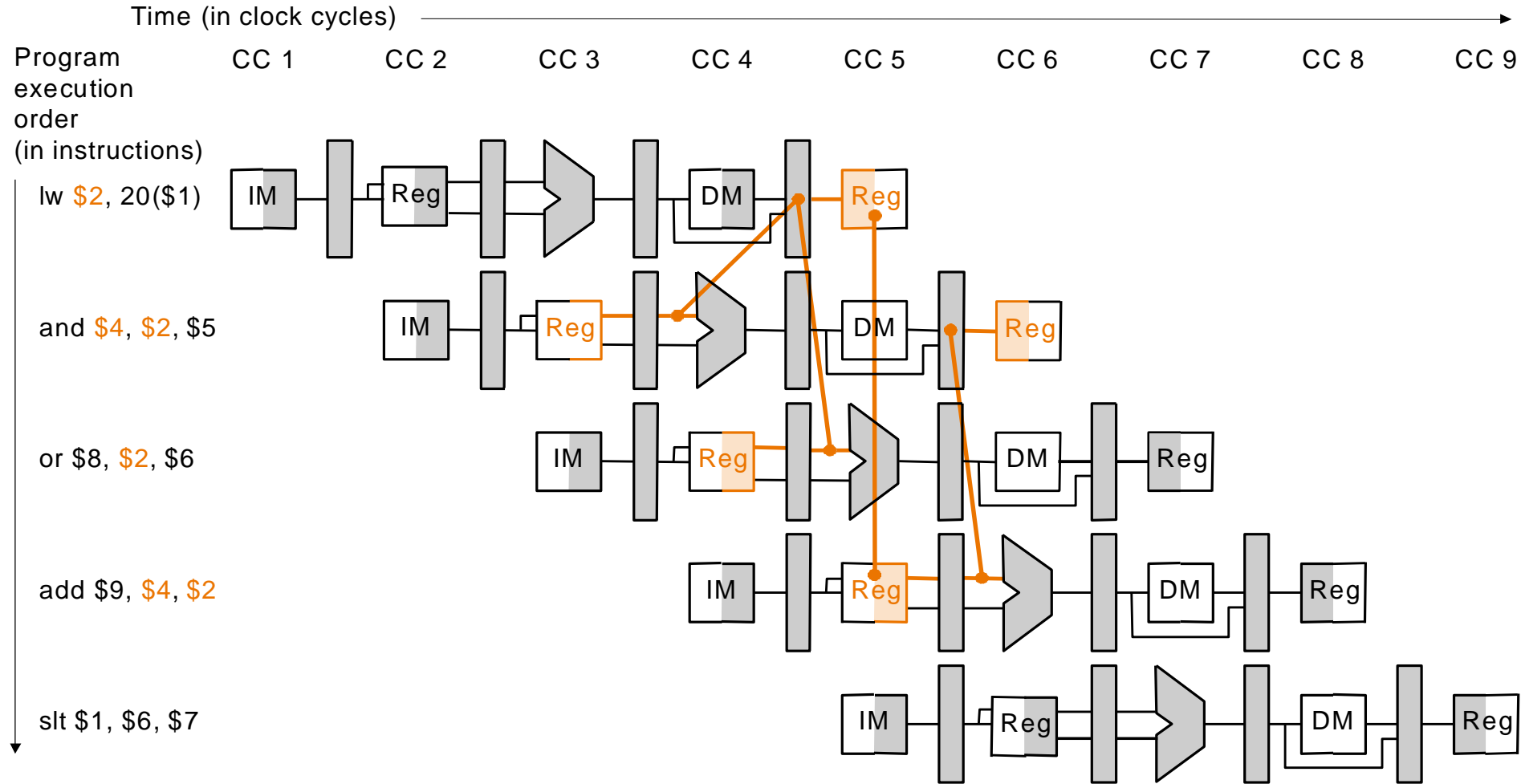
Clock 5



Example



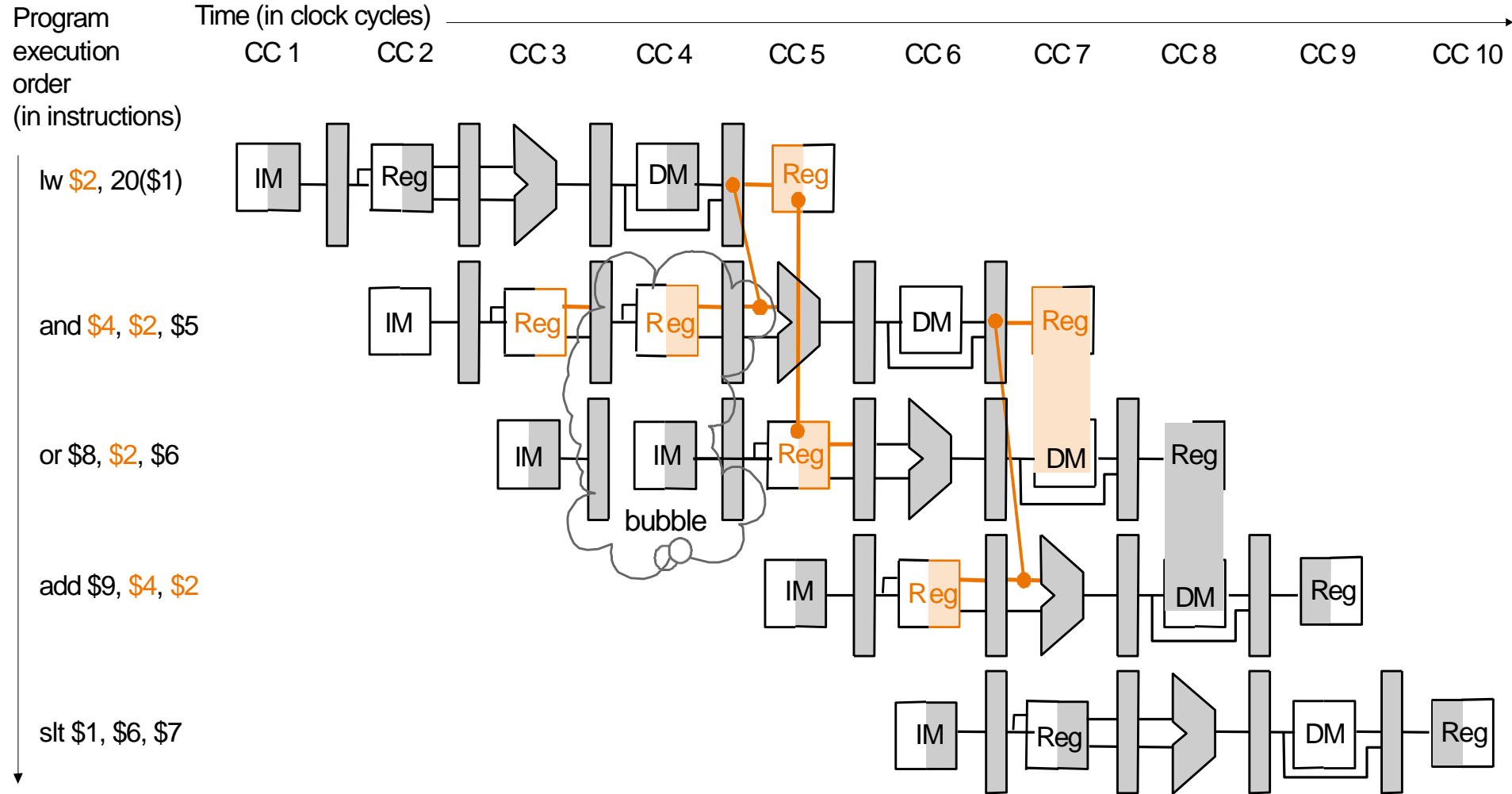
Data Hazard Caused by Load



- ❑ Dependencies backward in time are hazards
- ❑ Cannot solve with forwarding
- ❑ Must delay/stall instruction dependent on loads



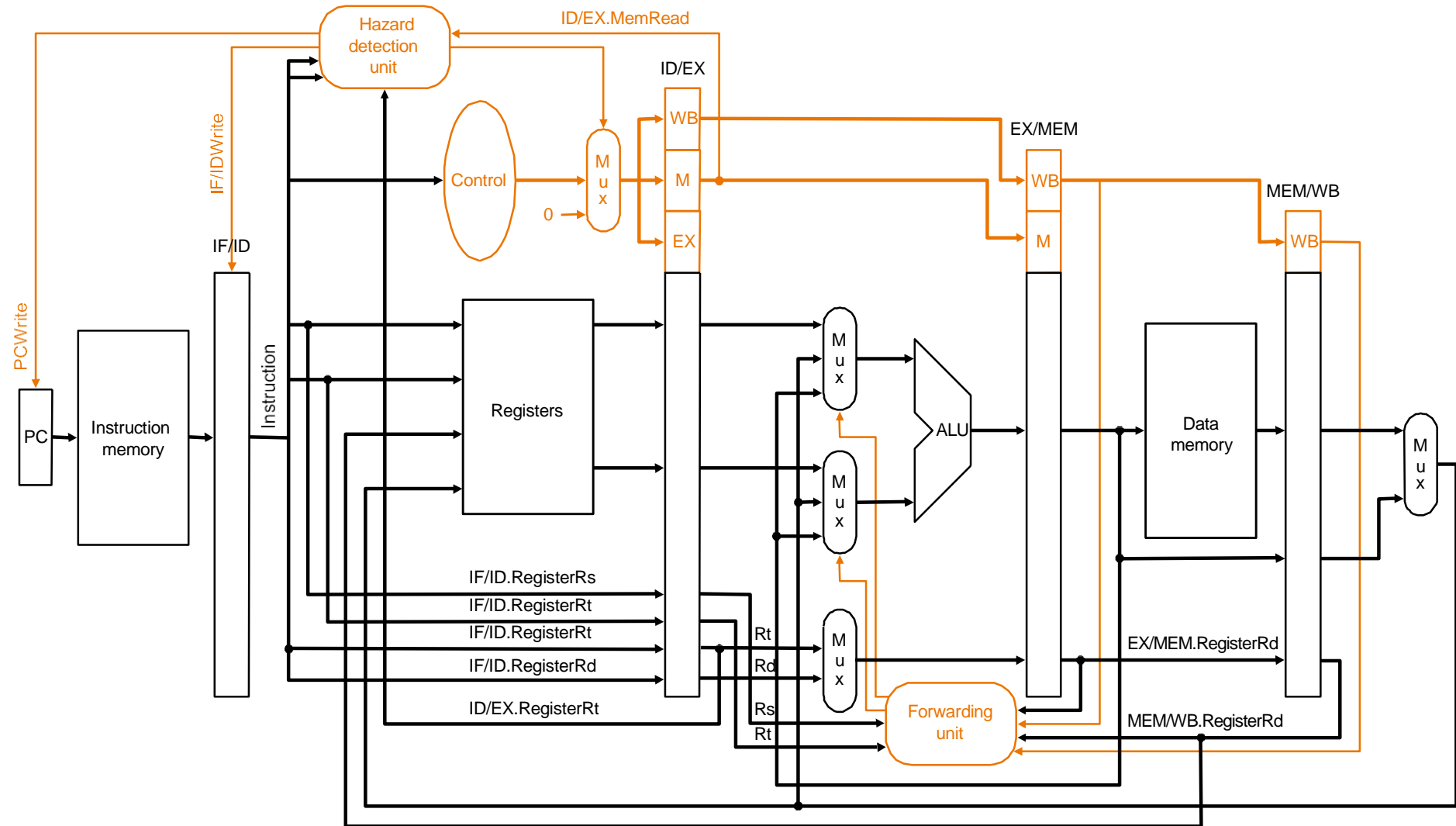
Hazard Detection Unit



if (ID/EX.MemRead & ((ID/EX.Register-Rt = IF/ID.Register-Rs) | (ID/EX.Register-Rt = IF/ID.Register-Rt))) stall the pipeline



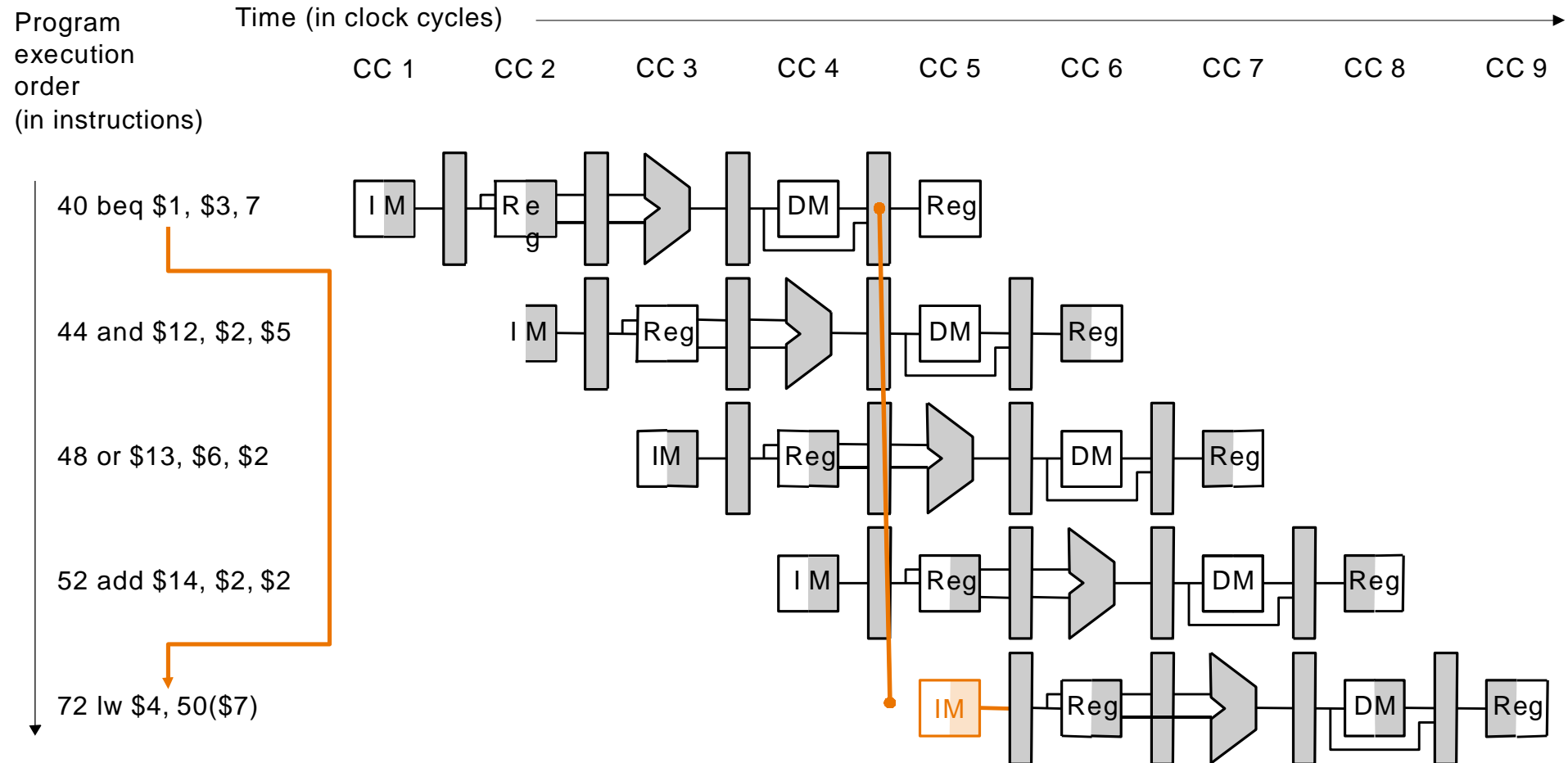
Supporting Pipeline Stall



De-asserting all the control lines in EX, MEM and WB control fields of the ID/EX pipeline stationary register would stall the pipeline

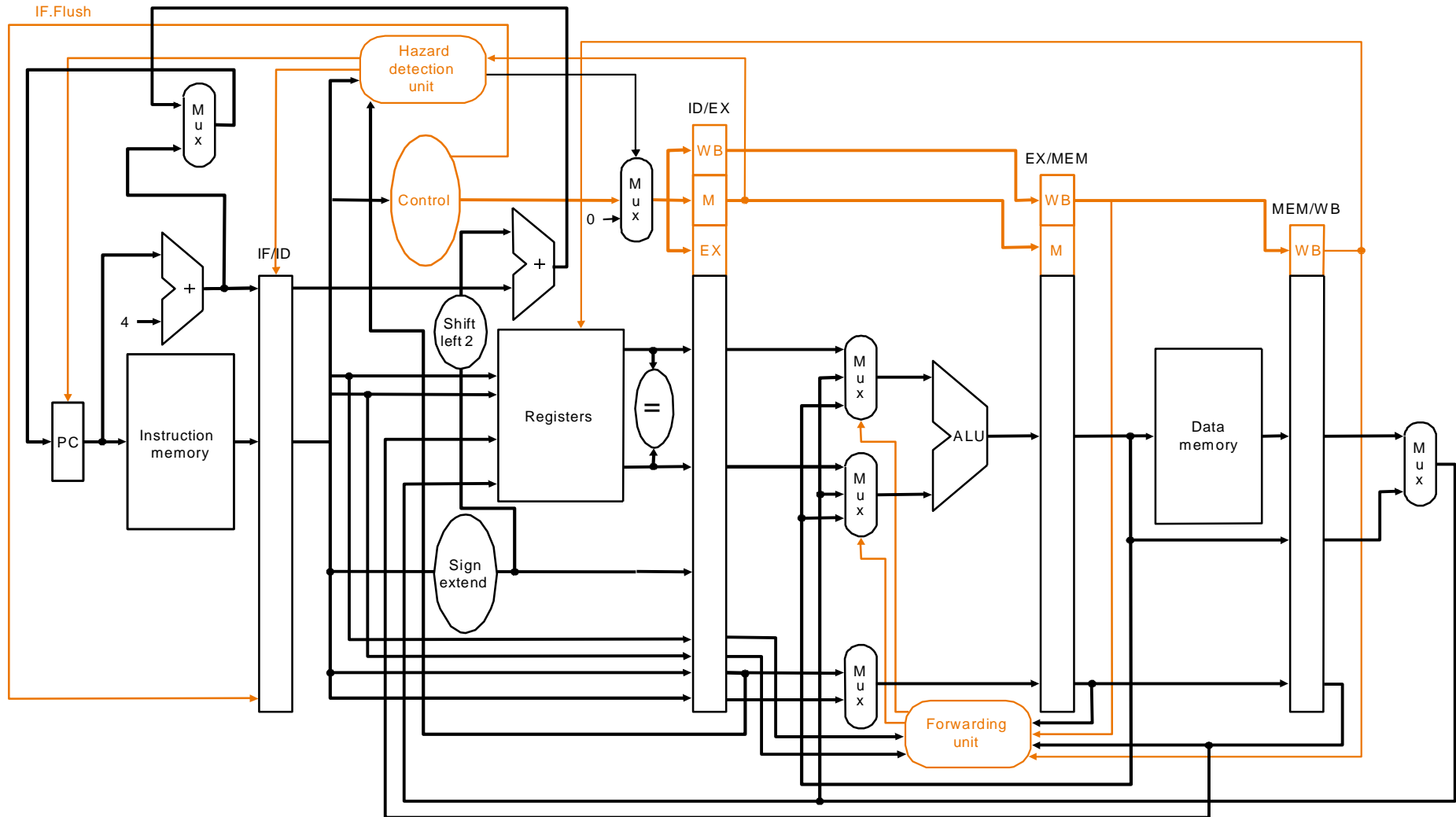


Control Hazards



- ❑ Must be carefully detected and handled to avoid semantic error
- ❑ Can be solved by excessive pipeline stalls which is inefficient
- ❑ Two approaches: “delayed branching” and “branch prediction”

Supporting Conditional Branching



- ❑ Moves comparison to ID stage to minimize worst case pipeline flushing
- ❑ Interacts with data hazards hardware to ensure right register values
- ❑ Flushes the pipeline with zero control values and instruction register



Conditional Branching Example

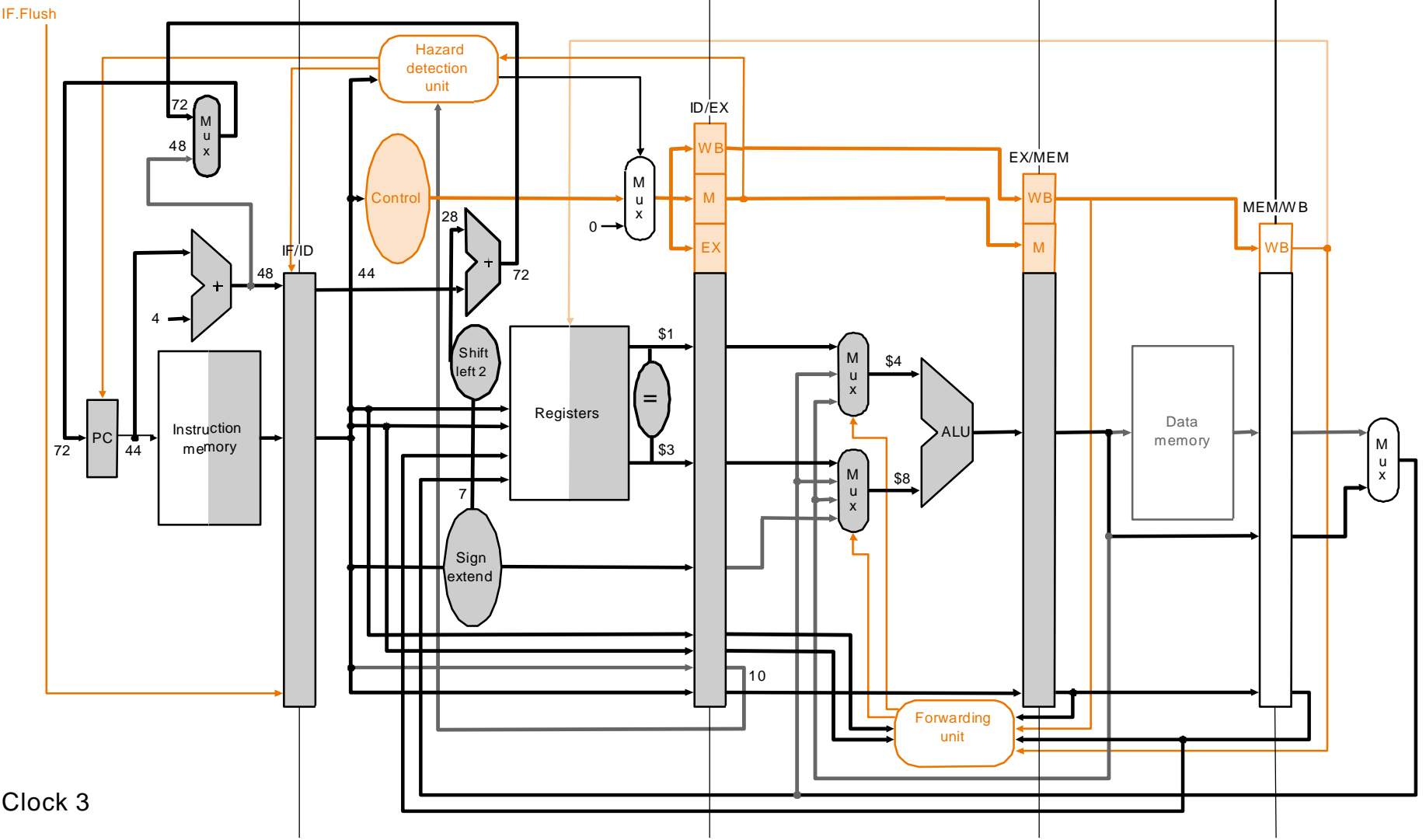
and \$12, \$2, \$5

beq \$1, \$3, 7

sub \$10, \$4, \$8

before<1>

before<2>

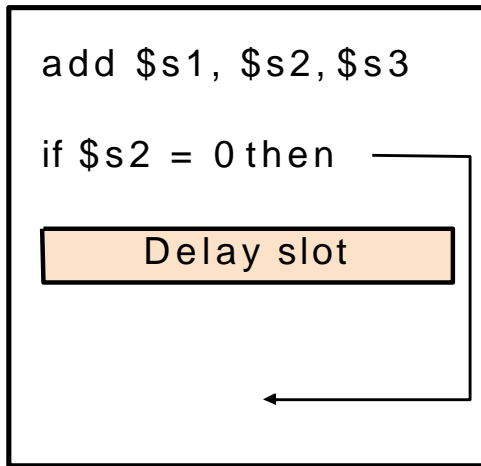


Clock 3

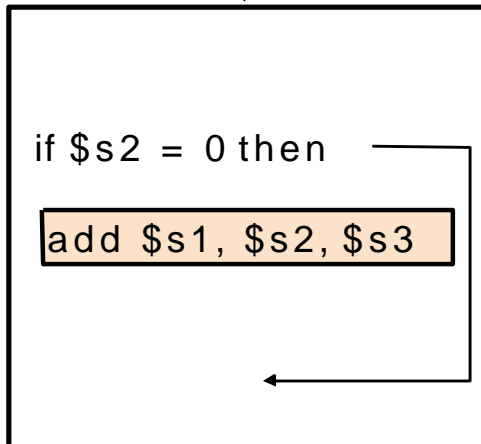


Compiler Optimization of Delays

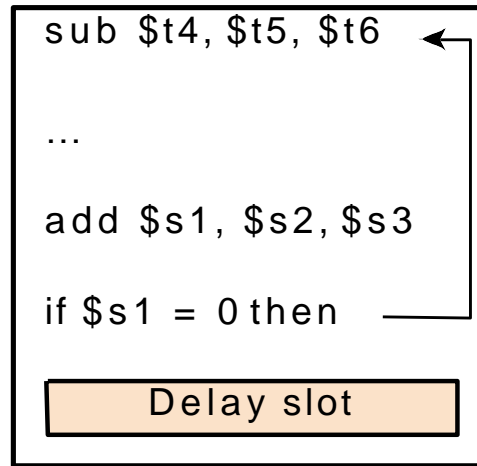
a. From before



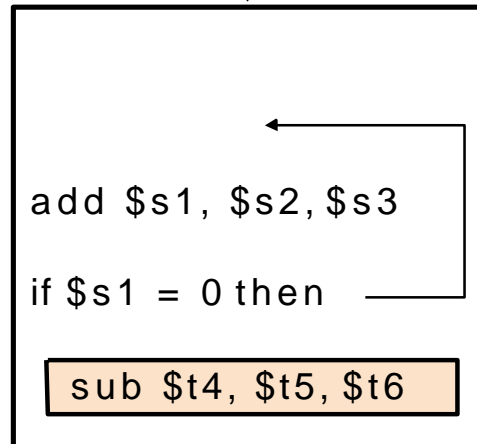
Becomes



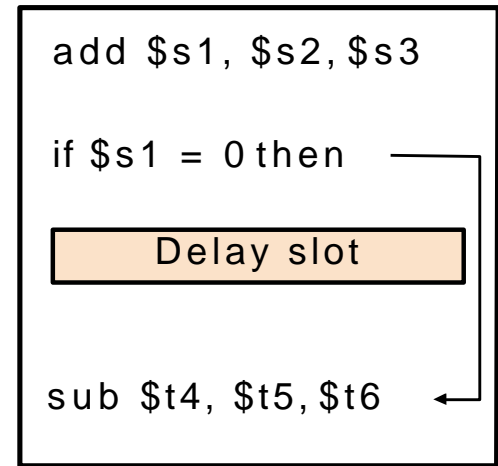
b. From target



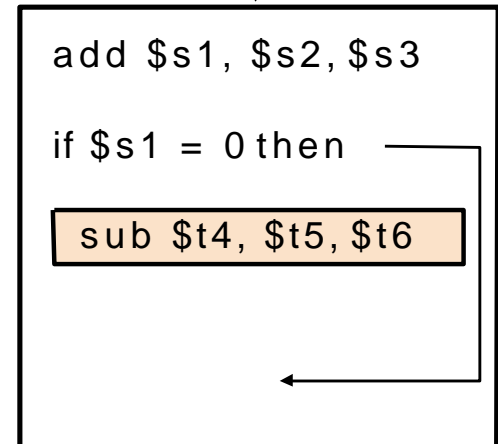
Becomes



c. From fall through



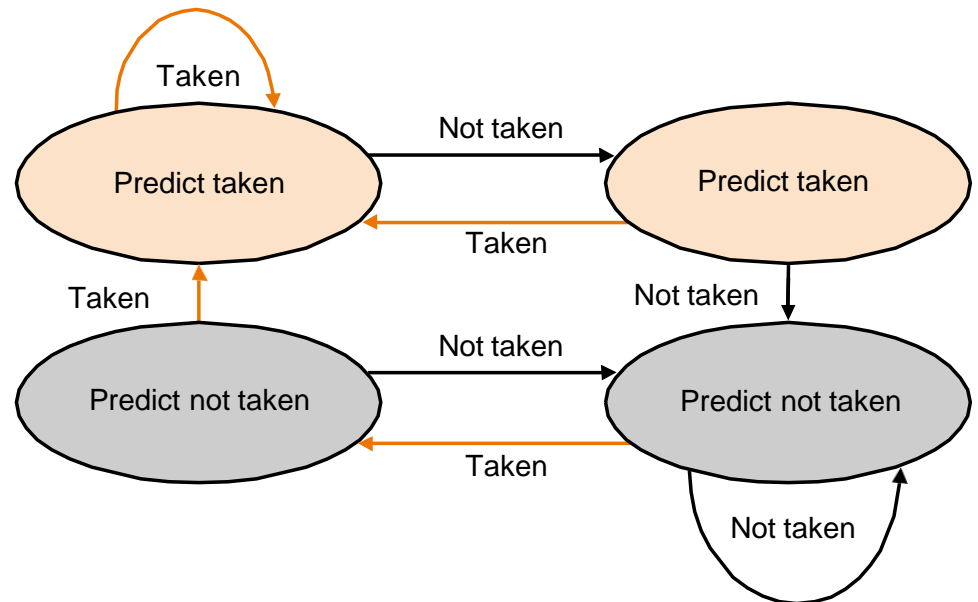
Becomes



Dynamic Branch Prediction

- ❑ Uses a history buffer to track branching instruction using their addresses
- ❑ Simplest approach uses one-bit per branch instruction to indicate the last experience
- ❑ Branching differently from prediction gets the history bit flipped
- ❑ One-bit buffer still fails to predict at least twice in case of a loop

A two-bit buffer better captures the history of the branch instruction



Exception/Interrupts

- ❑ 5 instructions executing in 5 stages pipeline
 - ➔ How to stop the pipeline?
 - ➔ Restart?
 - ➔ Who caused the interrupt?

Stage Problem interrupts occurring

IF Page fault on instruction fetch; memory-protection violation

ID Undefined or illegal op-code

EX Arithmetic exception

MEM Page fault on data fetch; memory-protection violation; memory error

- ❑ Load with data page fault, Add with instruction page fault?

- ❑ Solution: interrupt vector/instruction, interrupt ASAP, restart everything incomplete

❑ External Interrupts:

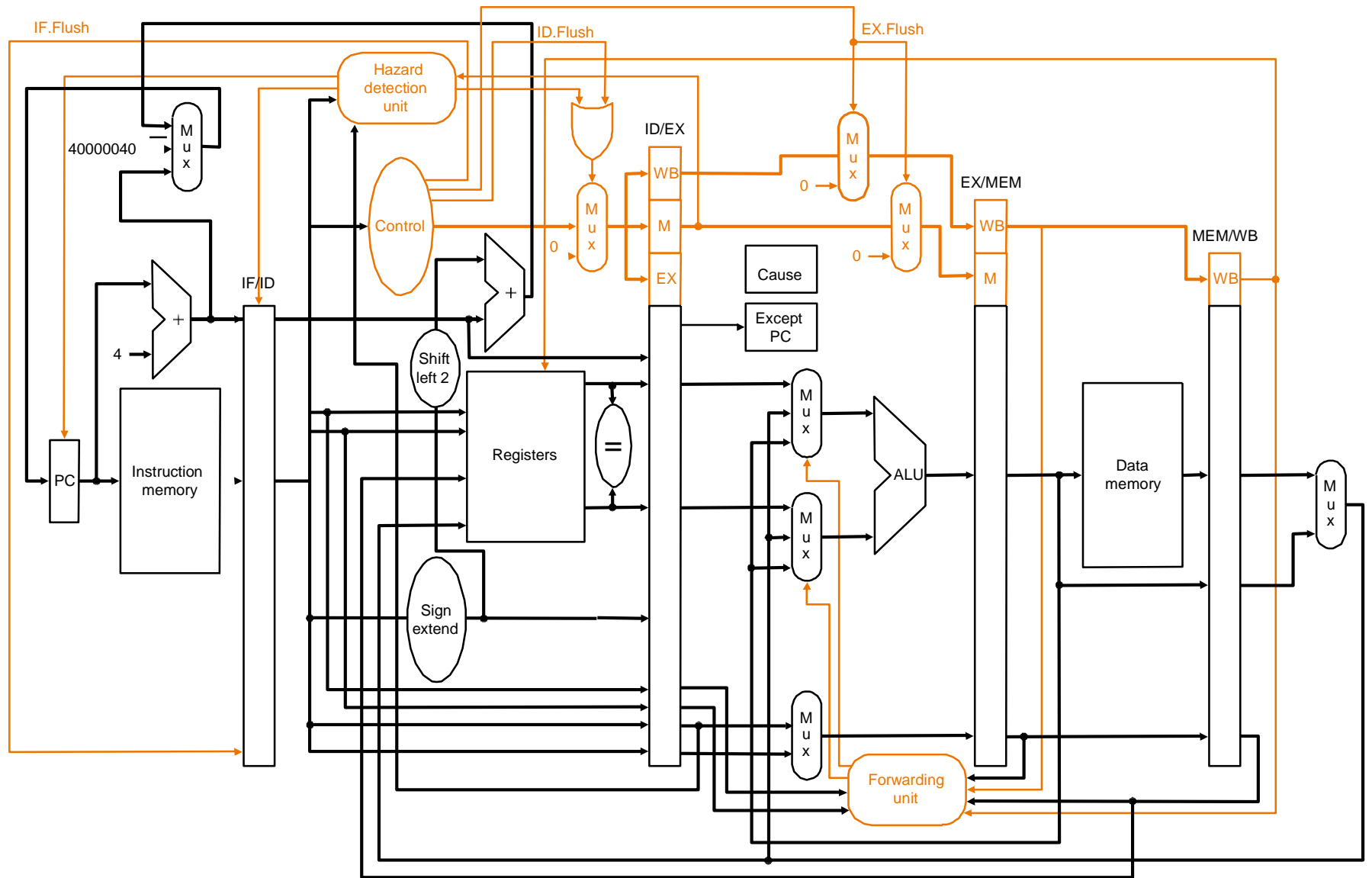
- ➔ Allow pipeline to drain
- ➔ Load PC with interrupt address

❑ Faults (within instr., can be restarted)

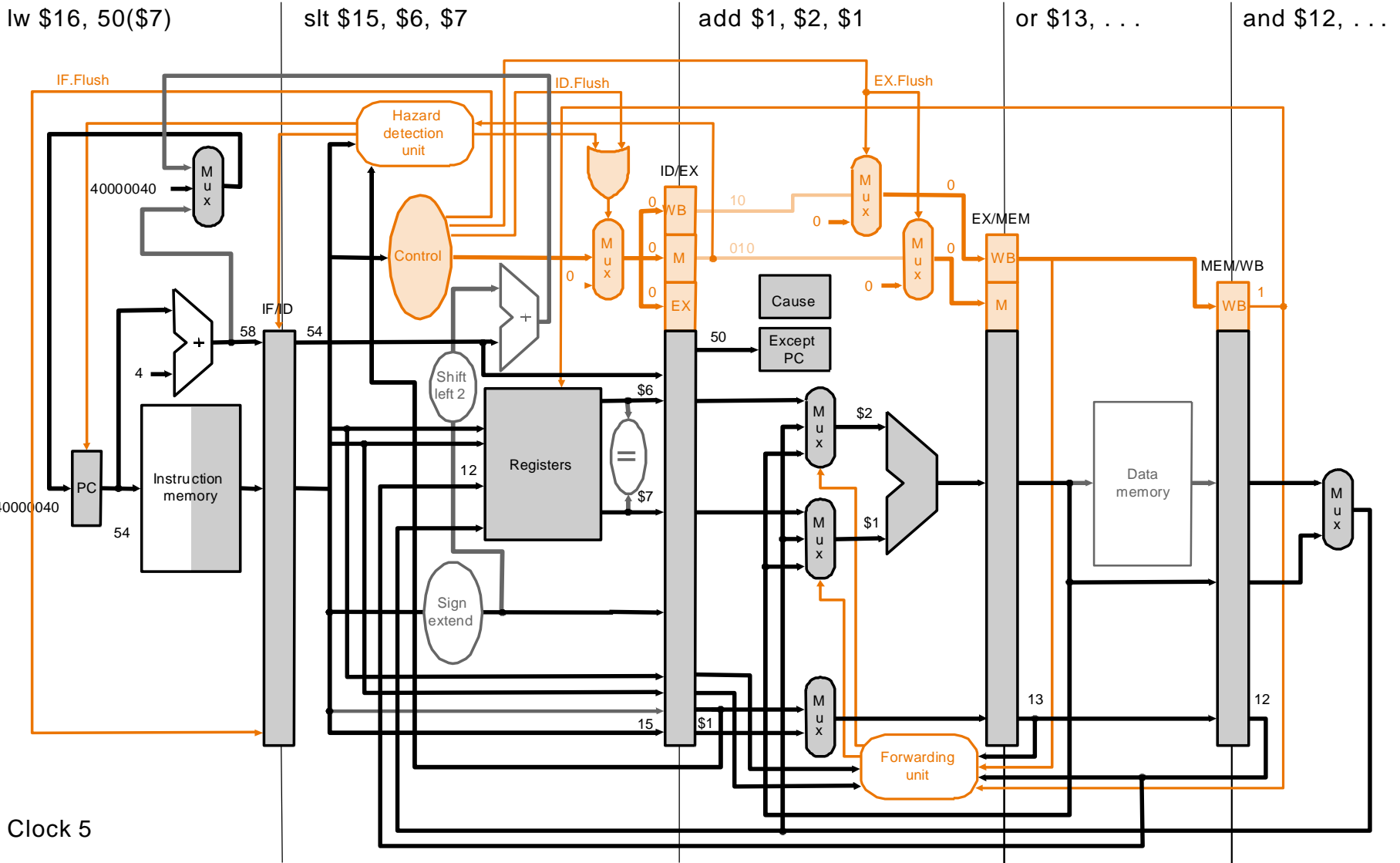
- ➔ Force trap instruction into IF
- ➔ Disable writes till trap hits WB
- ➔ Must save multiple PCs or PC + state



Supporting Exception Handling



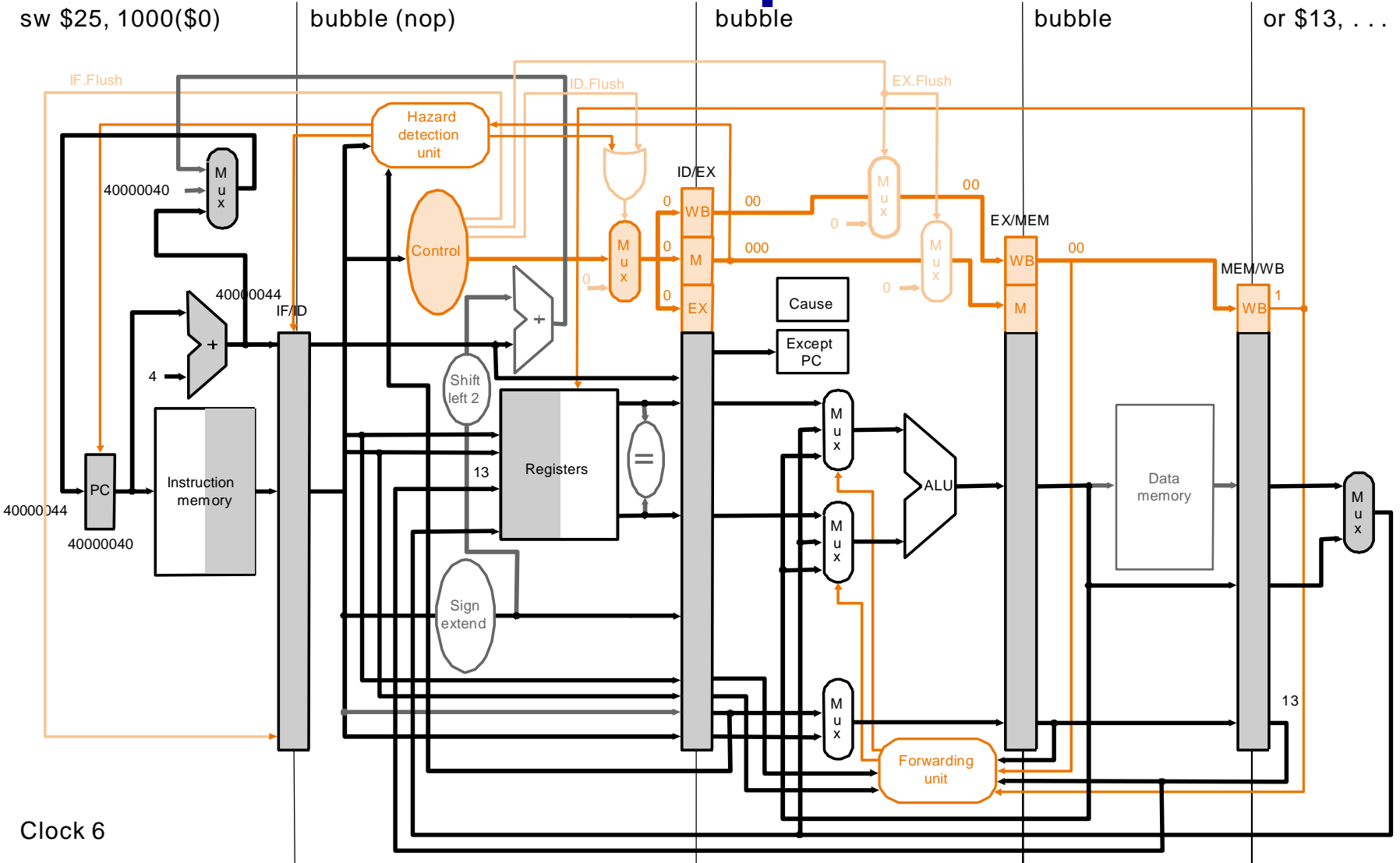
Example for Exception Handling



Add instruction causes overflow



Example



- Pipeline is drained after saving address of instruction
- First instruction of the handler is fetched



Conclusion

□ Summary

- Pipeline hazard detection
 - Data hazard
 - Control hazard
- Handling hazard in the pipeline design
 - Data forwarding and branch prediction
 - Coordination between control and data hazard
- Supporting exceptions
 - Pipeline flushing
 - Invoking exception handling routines

□ Next Lecture

- Pipelining floating point operations
- Instruction level parallelism

Read sections 4.7-4.9 in the 4th Ed or the 5th Ed. of the textbook

